下载APP

10 | 卷积（下）：计算机的眼睛

2021-11-01 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远
时长 14:19 大小 13.13M

你好，我是方远。

经过上一节课的学习，相信你已经对标准的卷积计算有所了解。虽然标准卷积基本上可以作为主力 Carry 全场，但是人们还是基于标准卷积，提出了一些其它的卷积方式，这些卷积方式在应对不同问题时能够发挥不同的作用，这里我为你列举了一些。

领资料

卷积类型	说明
深度可分离卷积	用于轻量化模型
空洞卷积	常用于图像分割
转置卷积	常用于图像分割
残差卷积块	为了提高网络精度的一种组合
Incetpion模块	为了提高网络精度的一种组合
SE 块	为了提高网络精度的一种组合

极客时间

在上一节课中，我们学习了 conv2d 的 in_channels、out_channels、kernel_size、stride、padding 与 bias 参数。

其中，PyTorch 中 conv2d 中剩余的两个参数，它们分别对应着两种不同的卷积，分别是深度可分离卷积和空洞卷积，让我们一起来看看。

深度可分离卷积（Depthwise Separable Convolution）

我们首先看看依托 groups 参数实现的深度可分离卷积。

随着深度学习技术的不断发展，许多很深、很宽的网络模型被提出，例如，VGG、ResNet、SENet、DenseNet 等，这些网络利用其复杂的结构，可以更加精确地提取出有用的信息。同时也伴随着硬件算力的不断增强，可以将这些复杂的模型直接部署在服务器端，在工业中可以落地的项目中都取得了非常优秀的效果。

但这些模型具有一个通病，就是速度较慢、参数量大，这两个问题使得这些模型无法被直接部署到移动终端上。而移动端的各种应用无疑是当今最火热的一个市场，这种情况下这些深而宽的复杂网络模型就不适用了。

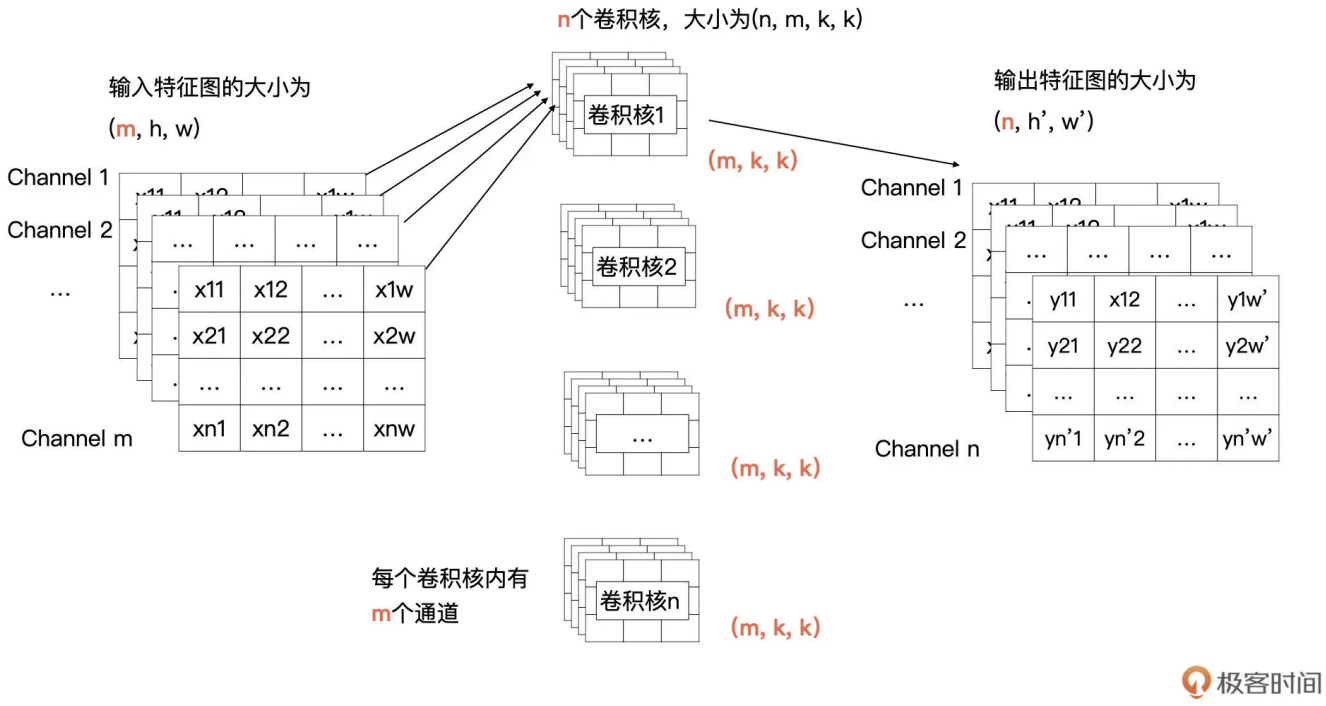
因此，很多研究将目光投入到寻求更加轻量化的模型当中，这些轻量化模型的要求是速度快、体积小，精度上允许比服务器端的模型稍微降低一些。

深度可分离卷积就是谷歌在 MobileNet v1 中提出的一种轻量化卷积。**简单来说，深度可分离卷积就是我们刚才所说的在效果近似相同的情况下，需要的计算量更少。**接下来，我们先来看看深度可分离卷积是如何计算的，然后再对比一下计算量到底减少了多少。

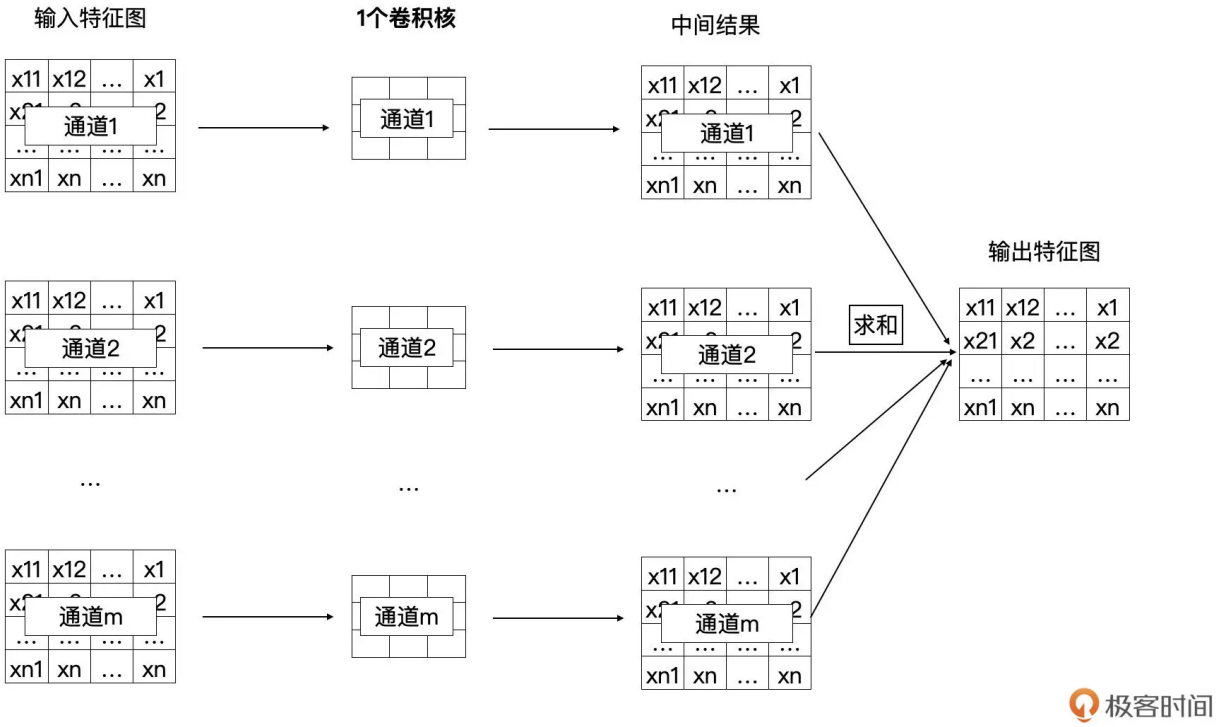
深度可分离卷积（Depthwise Separable Convolution）由 **Depthwise**（DW）和 **Pointwise**（PW）这两部分卷积组合而成的。

我们先来复习一下标准卷积，然后再来讲解一下获得同样输出特征图的深度可分离卷积是如何工作的。

你还记得下面这张图吗？这是我们上节课讲到的标准卷积计算方式，它描述的是：输入 m 个尺寸为 h, w 的特征图，通过卷积计算获得 n 个通道尺寸为 h' 与 w' 的特征图的计算过程。



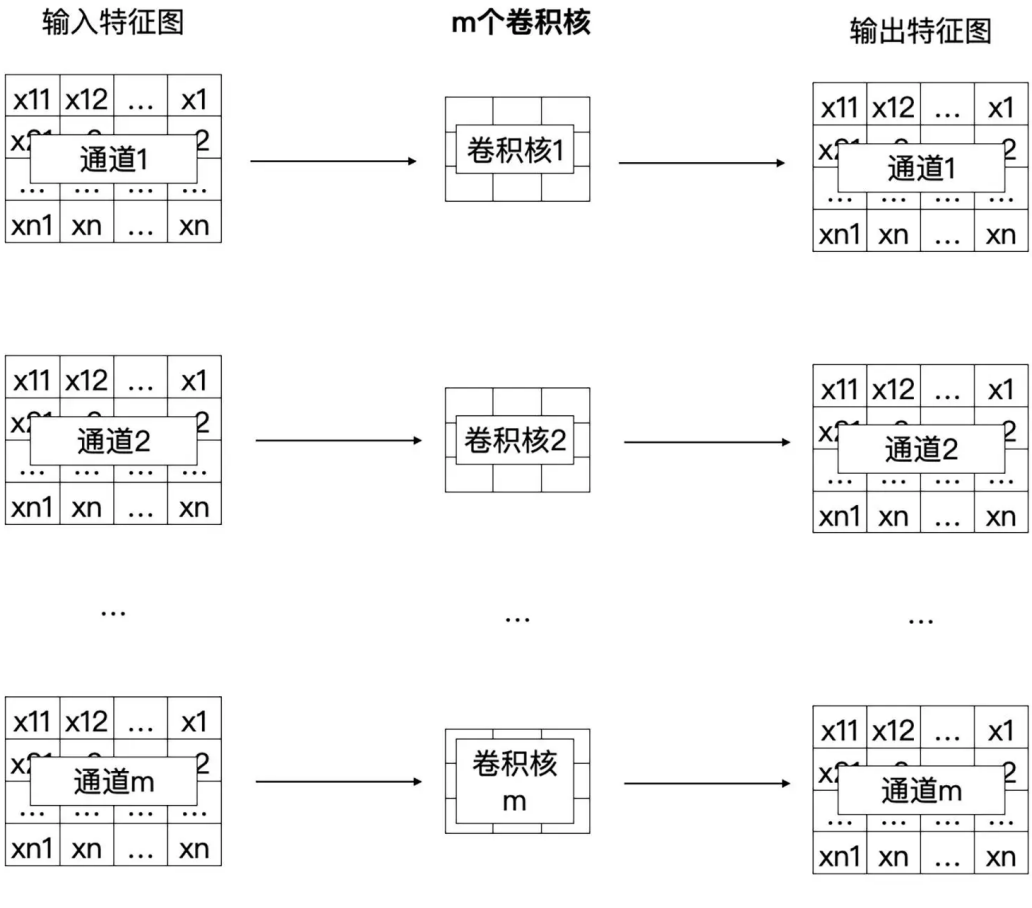
我们将特征图与一个卷积核计算的过程展开一下，请看下图。一个卷积核中的 m 个卷积分别与输入特征图的 m 个通道数据进行卷积计算，生成一个中间结果，然后 m 个中间结果按位求和，最终就能获得 n 个输出特征图中的一个特征图。



Depthwise（DW）卷积

那什么是 DW 卷积呢？DW 卷积就是有 m 个卷积核的卷积，每个卷积核中的通道数为 1，这 m 个卷积核分别与输入特征图对应的通道数据做卷积运算，所以 DW 卷积的输出是有 **m** 个通道的特征图。通常来说，DW 卷积核的大小是 3x3 的。

DW 卷积的过程如下图所示：



Pointwise（PW）卷积

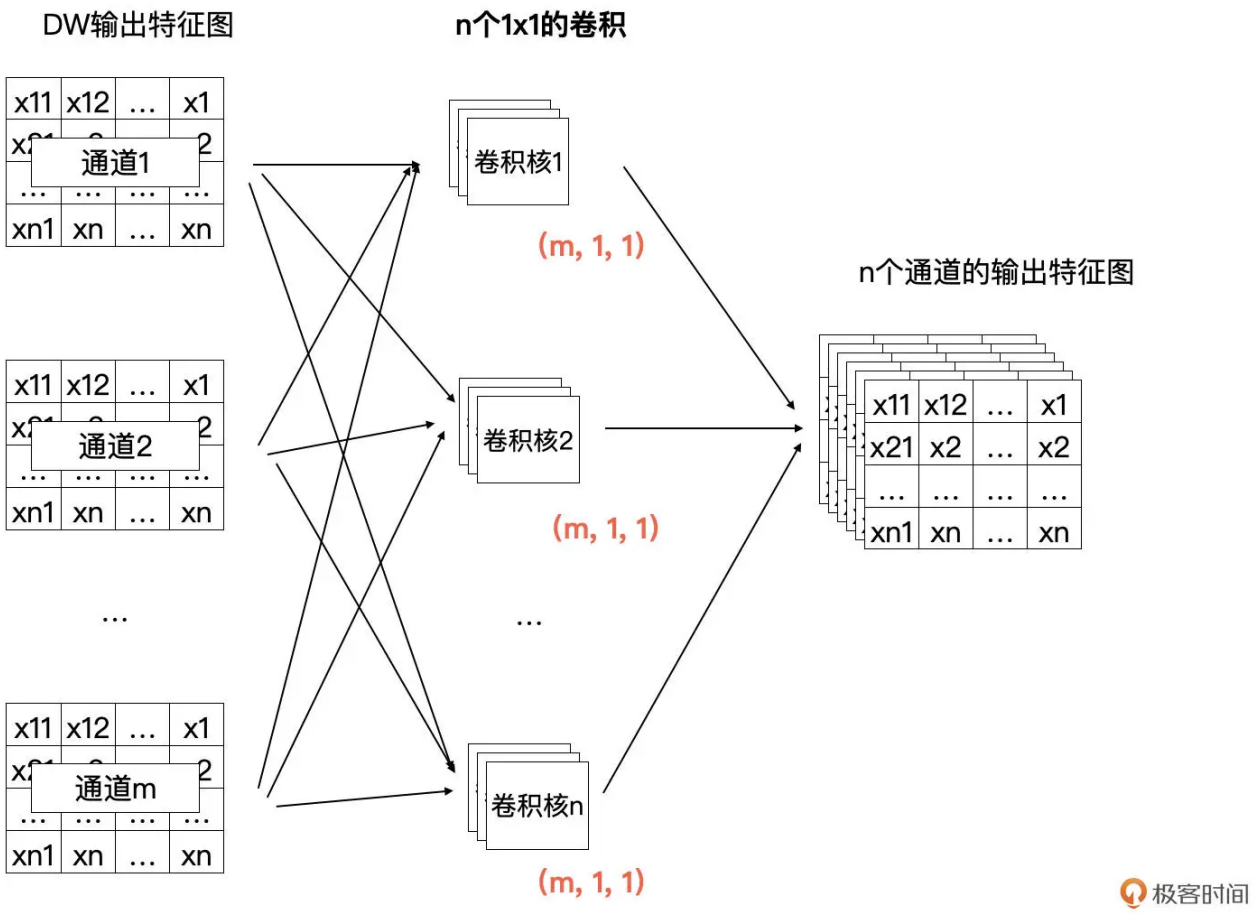
通常来说，深度可分离卷积的目标是轻量化标准卷积计算的，所以它是可以来替换标准卷积的，这也意味着原卷积计算的输出尺寸是什么样，替换后的输出尺寸也要保持一致。

所以，在深度可分离卷积中，我们最终要获得一个具有 n 个通道的输出特征图，而刚才介绍的 DW 卷积显然没有达到，并且 DW 卷积也忽略了输入特征图通道之间的信息。

所以，在 DW 之后我们还要加一个 PW 卷积。PW 卷积也叫做逐点卷积。PW 卷积的主要作用就是将 DW 输出的 m 个特征图结合在一起考虑，再输出一个具有 n 个通道的特征图。

在卷积神经网络中，我们经常可以看到使用 1x1 的卷积，1x1 的卷积主要作用就是升维与降维。所以，在 DW 的输出之后的 PW 卷积，就是 n 个卷积核的 1x1 的卷积，每个卷积核中有 m 个通道的卷积数据。

为了帮你理解刚才我描述的这个过程，我还是用图解的方式为你描述一下，你可以对照下图看一看：



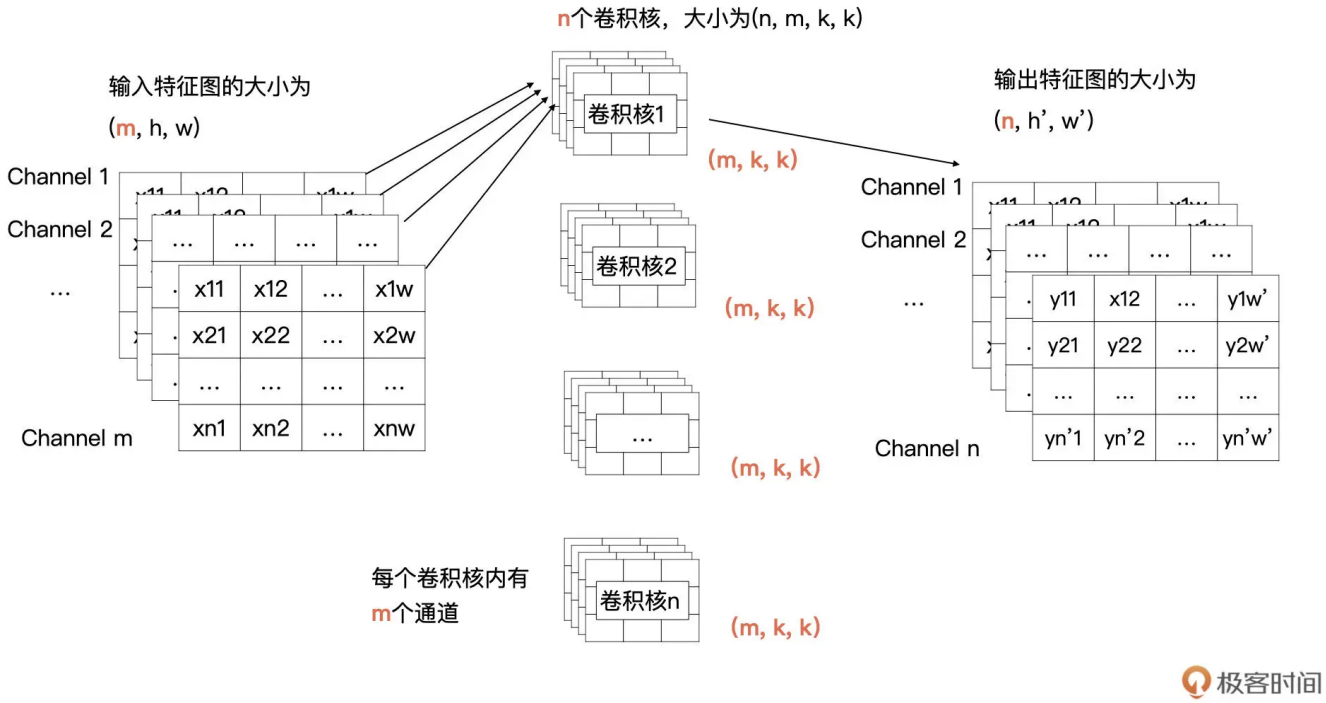
经过这样的 DW 与 PW 的组合，我们就可以获得一个与标准卷积有同样输出尺寸的轻量化卷积啦。既然是轻量化，那么我们下面就来看看，深度可分离卷积的计算量相对于标准卷积减少了多少呢？

计算量

我们的原问题是有 m 个通道的输入特征图，卷积核尺寸为 kxk，输出特征图的尺寸为 (n, h', w')，那么标准的卷积的计算量为：

$$k \times k \times m \times n \times h' \times w'$$

我们是怎么得出这个结果的呢？你可以从输出特征图往回思考。



上图输出特征图中每个点的数值是由 n 个卷积核与输入特征图计算出来的吧，这个计算量是 $k \times k \times m \times n$，那输出特征图有多少个点？没错，一共有 $h' \times w'$ 个。所以，我们自然就得出上面的计算方式了。

如果采用深度可分离卷积，DW 的计算量为：$k \times k \times m \times h' \times w'$，而 PW 的计算量为：$1 \times 1 \times m \times n \times h' \times w'$。

我们不难得出标准卷积与深度可分离卷积计算量的比值为：

$$\frac{k \times k \times m \times h' \times w' + 1 \times 1 \times m \times n \times h' \times w'}{k \times k \times m \times n \times h' \times w'}$$

$$= \frac{1}{n} + \frac{1}{k \times k}$$

所以，深度可分离卷积的计算量大约为普通卷积计算量的 $\frac{1}{k^2}$。

那深度可分离卷积落实到 PyTorch 中是怎么实现的呢？

PyTorch 中的实现

在 PyTorch 中实现深度可分离卷积的话，我们需要分别实现 DW 与 PW 两个卷积。我们先看看 DW 卷积，实现 DW 卷积的话，就会用到 nn.Conv2d 中的 groups 参数。groups 参数的作用就是控制输入特征图与输出特征图的分组情况。

当 groups 等于 1 的时候，就是我们上一节课讲的标准卷积，而 groups=1 也是 nn.Conv2d 的默认值。

当 groups 不等于 1 的时候，会将输入特征图分成 groups 个组，每个组都有自己对应的卷积核，然后分组卷积，获得的输出特征图也是有 groups 个分组的。需要注意的是，**groups 不为 1 的时候，groups 必须能整除 in_channels 和 out_channels**。

当 groups 等于 in_channels 时，就是我们的 DW 卷积啦。

好，下面我们一起动手操作一下，看看如何实现一个 DW 卷积。首先我们来生成一个三通道的 5x5 输入特征图，然后经过深度可分离卷积，输出一个 4 通道的特征图。

DW 卷积的实现代码如下：

复制代码

```
import torch
import torch.nn as nn

# 生成一个三通道的5x5特征图
x = torch.rand((3, 5, 5)).unsqueeze(0)
print(x.shape)
# 输出：
torch.Size([1, 3, 5, 5])
# 请注意DW中，输入特征通道数与输出通道数是一样的
in_channels_dw = x.shape[1]
out_channels_dw = x.shape[1]
# 一般来讲DW卷积的kernel size为3
kernel_size = 3
stride = 1
# DW卷积groups参数与输入通道数一样
dw = nn.Conv2d(in_channels_dw, out_channels_dw, kernel_size, stride, groups=in
```

你需要注意以下几点内容：

1.DW 中，输入特征通道数与输出通道数是一样的；

2. 一般来讲，DW 的卷积核为 3x3；

3.DW 卷积的 groups 参数与输出通道数是一样的。

好啦，DW 如何实现我们已经写好了，接下来就是 PW 卷积的实现。其实 PW 卷积的实现就是我们上一节课介绍的标准卷积，只不过卷积核为 1x1。需要注意的是，PW 卷积的 groups 就是默认值了。

具体代码如下所示：

复制代码

```
in_channels_pw = out_channels_dw
out_channels_pw = 4
kernel_size_pw = 1
pw = nn.Conv2d(in_channels_pw, out_channels_pw, kernel_size_pw, stride)
out = pw(dw(x))
print(out.shape)
```

好了，groups 以及深度可分离卷积就讲完了，接下来我们看看最后一个 dilation 参数，它是用来实现空洞卷积的。

空洞卷积

空洞卷积经常用于图像分割任务当中。图像分割任务的目的是要做到 pixel-wise 的输出，也就是说，对于图片中的每一个像素点，模型都要进行预测。

对于一个图像分割模型，通常会采用多层卷积来提取特征的，随着层数的不断加深，感受野也越来越大。这里有个新名词——“感受野”，这个我稍后再解释。我们先把空洞卷积的作用说完。

但是对于图像分割模型有个问题，经过多层的卷积与 pooling 操作之后，特征图会变小。为了做到每个像素点都有预测输出，我们需要对较小的特征图进行上采样或反卷积，将特征图扩大到一定尺度，然后再进行预测。

要知道，从一个较小的特征图恢复到一个较大的特征图，这显然会带来一定的信息损失，特别是较小的物体，这种损失是很难恢复的。那问题来了，能不能既保证有比较大的感受

野，同时又不用缩小特征图呢？

估计你已经猜到了，空洞卷积就是解决这个问题的杀手锏，它最大的优点就是不需要缩小特征图，也可以获得更大的感受野。

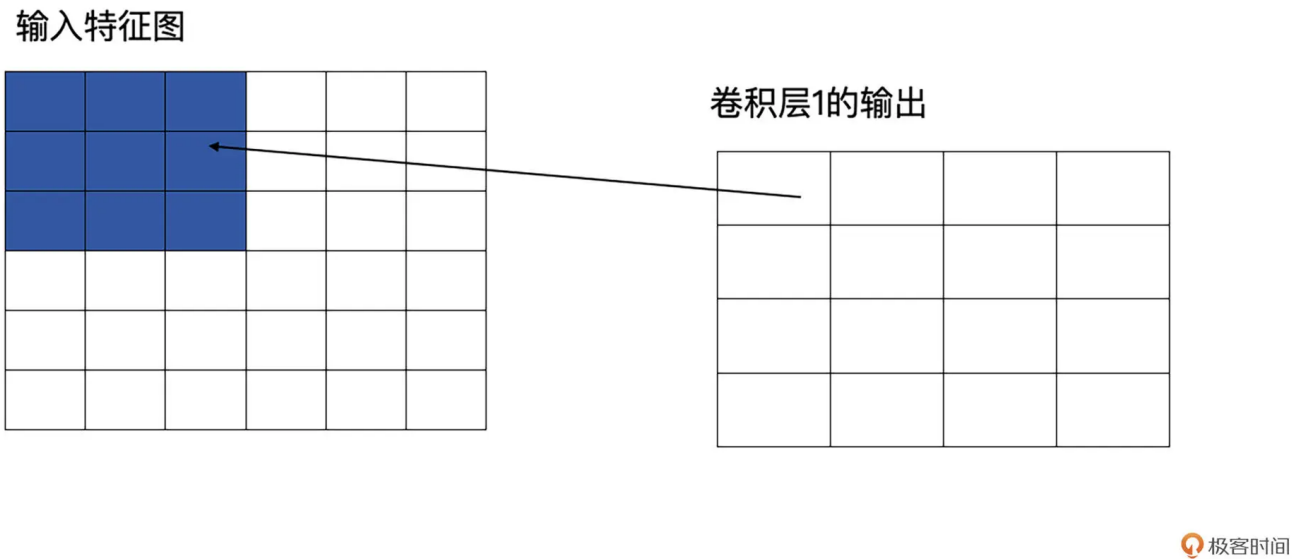
感受野

现在让我来解释一下什么是感受野。感受野是计算机视觉领域中经常会看到的一个概念。

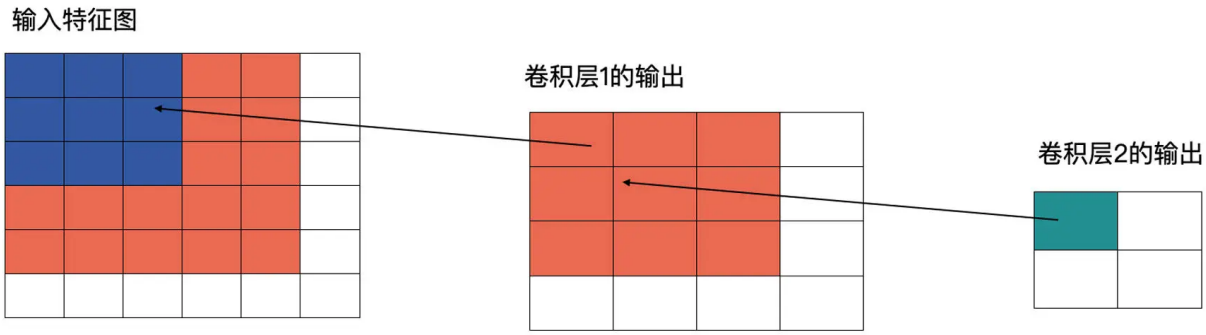
因为伴随着不断的 pooling（这是卷积神经网络中的一种操作，通常是在一定区域的特征图内取最大值或平均值，用最大值或平均值代替这个区域的所有数据，pooling 操作会使特征图变小）或者卷积操作，在卷积神经网络中不同层的特征图是越来越小的。

这就意味着在卷积神经网络中，相对于原图来说，不同层的特征图，其计算区域是不一样的，这个区域就是感受野。感受野越大，代表着包含的信息更加全面、语义信息更加抽象，而感受野越小，则代表着包含更加细节的语义信息。

光说理论不容易理解，我们还是结合例子看一看。请看下图，原图是 5x5 的图像，第一层卷积层为 3x3，这时它输出的感受野就是 3，因为输出的特征图中每个值都是由原图中 3x3 个区域计算而来的。



再看下图，卷积层 2 也为 3x3 的卷积，输出为 2x2 的特征图。这时卷积层 2 的感受野就会变为 5（输入特征图中蓝色加橘黄色部分）。



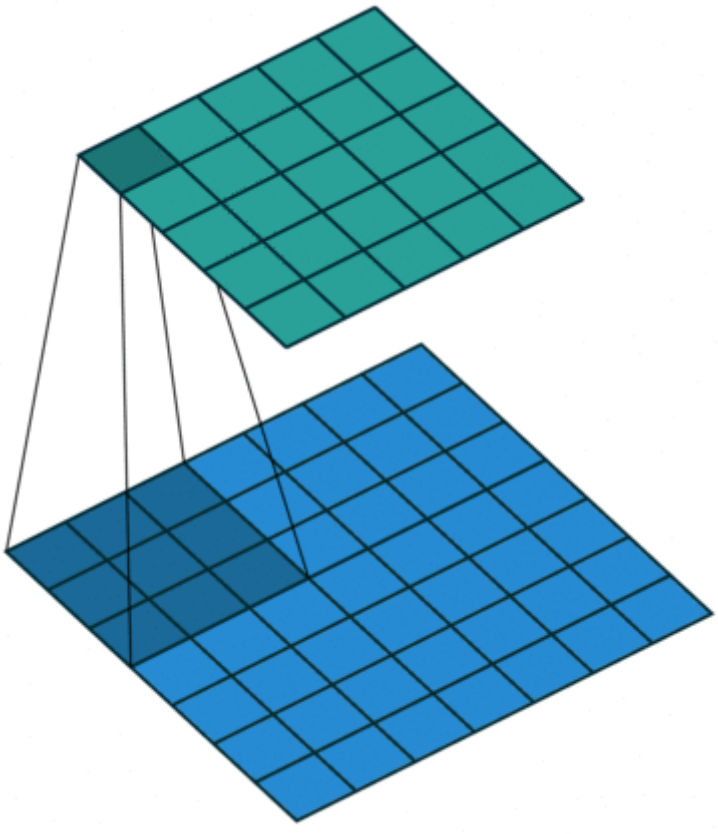
配合图解，我相信你很容易就能明白感受野的含义了。

计算方式

好，那么我们再来看看空洞卷积具体是如何计算的。

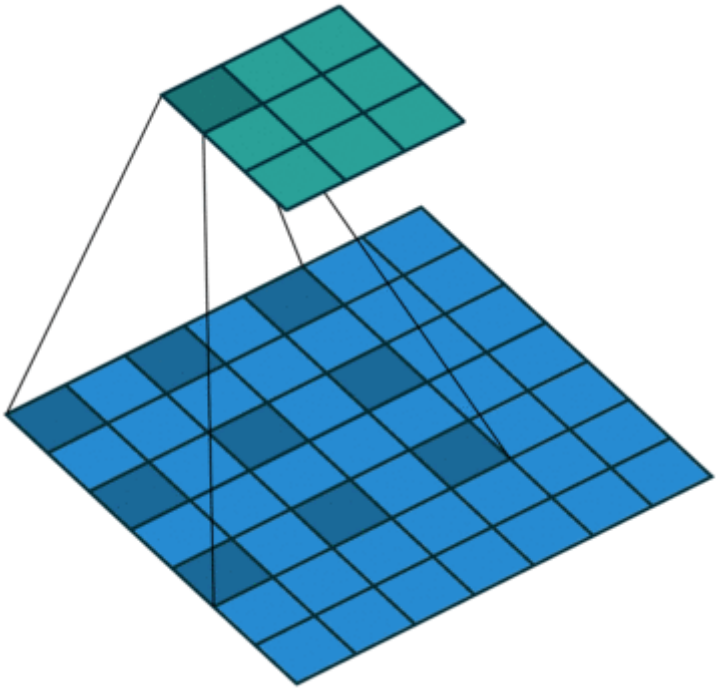
用语言来描述空洞卷积的计算方式比较抽象，我们不妨看一下它的动态示意图（这个GitHub中有各种卷积计算的动态图，非常直观，我们借助它来学习一下空洞卷积）。

首先，我们先来看看上节课讲的标准卷积是如何计算的。



对照上图，下面的蓝图为输入特征图，滑动的阴影为卷积核，绿色的为输出特征图。

然后我们再对照一下的空洞卷积示意图。



结合示意图我们会发现，计算方式与普通卷积一样，只不过是将卷积核以一定比例拆分开来。实现起来呢，就是用 0 来充填卷积核。

这个分开的比例，我们一般称之为扩张率，就是 Conv2d 中的 dilation 参数。

dilation 参数默认为 1，同样也是可以为 int 或者 tuple。当为 tuple 时，第一位代表行的信息，第二位代表列的信息。

总结

恭喜你，完成了今天的学习。今天我们在实现 PyTorch 卷积操作的同时，学习了两个特殊的卷积，深度可分离卷积与空洞卷积。

对于空洞卷积，你最需要掌握的是感受野这个概念，以及空洞卷积的计算方式。感受野就是能在原始图像中反应的区域。

深度可分离卷积主要用于轻量化的模型，而空洞卷积主要用于图像分割任务中。

这里分享一下我的经验：如果说你需要轻量化你的模型，让你的模型变得更小、更快，你可以考虑将卷积层替换为深度可分离卷积。如果你在做图像分割项目的话，可以考虑将网络靠后的层替换为空洞卷积，看看效果是否能有所提高。

最后，我们再来总结一下 PyTorch 中卷积操作的各个重要参数，我用表格的方式帮你做了总结归纳，你可以把它作为自己的工具包，时常翻看。

卷积类型	说明
in_channels	输入特征图中的通道数
out_channels	输出特征图的通道数，也就是卷积核数
kernel_size	卷积核的尺寸
stride	卷积核滑动的步长，如果padding方式为'valid'或'same'，则stride必须为1
padding	padding参数
dilation	空洞率
groups	将输入特征图分组进行计算。深度可分离卷积中，该参数等于输入特征图数
bias	是否加偏移项，一般都加

极客时间

每课一练

随机生成一个 3 通道的 128x128 的特征图，然后创建一个有 10 个卷积核且卷积核尺寸为 3x3（DW 卷积）的深度可分离卷积，对输入数据进行卷积计算。

欢迎你在留言区跟我交流互动，也推荐你把今天的内容分享给更多同事、朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 卷积（上）：计算机的眼睛

下一篇 11 | 损失函数：如何帮助模型学会“自省”？

精选留言 (3)

写留言

IUniverse
2021-11-01

x=torch.rand((3,128,128)).unsqueeze(0)
print(x.shape)

in_channel_DW=x.shape[1]
out_channel_DW=x.shape[1]...
展开

作者回复: 你好，IUniverse。感谢你的留言。正确👍👍👍，^^。

1 1

Geek_a95f0e
2021-11-04

import torch

```
input_feat=torch.randint(0,100,(1,3,128,128),dtype=torch.float32)
dw=torch.nn.Conv2d(3,3,3,groups=3)
pw=torch.nn.Conv2d(3,10,1)
result=pw(dw(input_feat))…
```

展开

作者回复: 你好，谢谢你的留言。正确👍^^

vcjmhg
2021-11-03

```
import torch
import torch.nn as nn

# 生成一个三通道的128x128特征图
x = torch.rand((3, 128, 128)).unsqueeze(0)…
```

展开

作者回复: 👍👍👍👍👍👍^^。